リングの方向付け問題を有限状態数で解く自己安定アルゴリズム(アルゴリズムと計算量理論)

AUTHOR(S):

梅本, 成俊; 角川, 裕次; 山下, 雅史

CITATION:

梅本, 成俊 ...[et al]. リングの方向付け問題を有限状態数で解く自己安定アルゴリズム(アルゴリズムと計算量理論). 数理解析研究所講究録 1995, 906: 257-263

ISSUE DATE:

1995-04

URL:

http://hdl.handle.net/2433/59435

数理解析研究所講究録
906巻 1995年 257-263

リングの方向付け問題を有限状態数で解く自己安定アルゴリズム

広島大学大学院工学研究科情報工学専攻　梅本成俊　(Narutoshi Umemoto)
広島大学工学部第二類　角川裕次　(Hirotsugu Kakugawa)
広島大学工学部第二類　山下雅史　(Masafumi Yamashita)

Abstract

自己安定アルゴリズム (self-stabilizing algorithm) は任意の初期状況から実行を開始することができ、また実行中に一過性の故障 (transient failure) が生じても有限時間内に問題を解くことができる分散アルゴリズムであり、フォールトトレランスに対する1つのアプローチとして多くの研究がなされている。Israeli らは均一な偶数サイズのリングネットワークにおいてDデーモン・R/Wデーモンの下では方向付け問題が解けないことを示し、さらにDデーモンの下で定数状態数で方向付け問題を解く確率アルゴリズムを提案した。また、片山らはこの問題を解く決定性アルゴリズムについて考察を行ない、リングが奇数サイズであればDデーモンの下でこの問題を無限状態数で解く決定性アルゴリズムが存在することを示した。本稿では、奇数サイズのリングネットワークにおいてDデーモンの下で状態数6で解く決定性アルゴリズムを提案し、その正当性の証明を行なう。

1 はじめに

複数の計算機 (以下プロセッサ) とその間を接続する通信リンクにより構成されるシステムを分散システムという。分散システムにおいてフォールトトレランスを実現するための一つのアプローチとして、1974年E.W.Dijkstra は自己安定システム (self-stabilizing system) を提案した[6]。自己安定システムは、次の2つの重要な長所を有している。

1. ネットワーク状況をある特定の状況に初期化する必要がなく、任意の初期状況からシステムを起動することができる。

2. 任意の状況からシステムを正当な状況 (対象とする問題によって定まる) に遷移させることができ、システムの作動中にデータの損失・変質といった (局所的には検出できない) 一過性の故障 (transient failure) が生じても外的補助なしに問題を解くことができる。

自己安定の概念は、もともと相互排除問題 (mutual exclusion problem) を対象として考えられ、以前はこの問題に対するアルゴリズムが多く提案された[1][6]。しかし、近年ではリングの方向付け問題 (ring orientation problem)[3][5] や、リーダー選挙問題 (leader election problem)[4] など様々な問題に対するアルゴリズムが提案されており、現在も活発な研究が行なわれている。これまでに提案された自己安定アルゴリズムは、主にプロセッサ間の通信方法・原子動作の大きさ・プロセッサの均一性により分類される。

1. プロセッサ間の通信

   - メッセージ通信モデル
     各プロセッサはメッセージを送受信することにより通信を行なう。
   - 状態通信モデル
     各プロセッサは隣接プロセッサの状態を直接知ることができる。
   - レジスタ通信モデル
     各プロセッサは隣接プロセッサに対する通信リンクごとに用意された通信用のレジスタにメッセージを書き込むことにより通信を行なう。

2. 原子動作

   - Cデーモン (central daemon)
     1単位時間に同時に動作可能なプロセッサはただ1つであり、かつ1回の動作で全隣接レジスタからの読み込み、状態の変更および必要であれば全隣接レジスタへの書き込みを行なうことができる。
   - Dデーモン (distributed daemon) 1単位時間に同時に複数のプロセッサが動作可能であることを除き、Cデーモンと同様の動作をする。

- R/W デーモン (read/write daemon)
  1単位時間に同時に動作可能なプロセッサはただ1つであり、かつ1回の動作で1回の隣接レジスタへの書き込みおよび状態の変更、または1回の隣接レジスタからの読み込みおよび状態の変更を行なうことができる。

3. プロセッサの均一性
   ネットワークを構成する全てのプロセッサは同一のアルゴリズムを実行し、また互いに区別することのできる識別子を持たない。このようなネットワークは均一 (uniform) であるという。

本稿ではリングの方向付け問題を取り扱う。リングの方向付け問題とは、共通の方向感覚 (右または左) を持たないプロセッサ群に対して、共通の方向感覚を与える問題である。この問題を解く自己安定アルゴリズムについて、これまでに偶数サイズの均一なリングネットワークにおいて、Dデーモン・R/Wデーモンの下で方向付け問題を解く決定性アルゴリズムは存在しないこと [5] や、奇数サイズのリングネットワークにおいて、Dデーモンの下で方向付け問題を無限状態数で解く決定性アルゴリズムが存在すること [3] などが知られている。状態数が無限であるアルゴリズムは、アルゴリズム的観点からすれば興味深いかもしれないが、実際上の立場からすれば、そのようなアルゴリズムはあまり意味がない。そこで、本稿では奇数サイズのリングネットワークにおいて、Dデーモンの下で方向付け問題を状態数6 (定数状態数) で解く決定性アルゴリズムを示し、その正当性の証明を行なう。

2 ネットワークモデル

2.1 ネットワークに関する諸定義

定義1 (ネットワーク) ネットワークは n 個 (n は奇数) のプロセッサ $P_0, P_1, \cdots, P_{n-1}$ が通信リンクで接続されたリングネットワークである。ただし、添字 $0, 1, \cdots, n-1$ は記述を容易にするためにのみ用いられ、アルゴリズム中で添字を利用することはない。各プロセッサは状態機械であるが、便宜上その動作をプログラムの形で与える。各プロセッサ P_i は P_{i-1} および P_{i+1} との間に通信リンクを持ち、P_{i-1}、P_{i+1} を P_i の隣接プロセッサという。また、これらのリンクに対して一方には1、他方には2が任意に番号が付けされており (この番号付けは終始変化しない)、P_i と1の番号付けがされたリンクで接続されたプロセッサを P_i の第1隣接プロセッサ、他方を P_i の第2隣接プロセッサという。 □

定義2 (プロセッサ間の通信) P_j を P_i の任意の1つの隣接プロセッサとする。このとき P_i と P_j との間の通信リンクは2個の通信用レジスタ R_{ij} および R_{ji} により構成される。すなわち、レジスタ通信モデルを採用する。P_i から P_j への通信は、P_i が R_{ij} に書き込んだ値を P_j が読み出すことにより行なわれる。逆方向の通信も同様である。 □

定義3 (プロセッサの均一性) ネットワークを構成する全てのプロセッサは同一のアルゴリズムを実行し、また互いに区別することのできる識別子を持たない。このようなネットワークは均一 (uniform) であるという。 □

定義4 (原子動作) 本稿ではDデーモンを仮定している。すなわち、1単位時間に複数のプロセッサが同時に動作することが可能であり、かつ1回の動作で、両隣接レジスタからの読み込み、状態の変更、さらに必要であれば両隣接レジスタへの書き込みを行なうことができる。 □

2.2 ネットワークに関する記述

定義5 (ネットワーク状況) ネットワークの状況 c を次のように定義する。

$$c = (s_0, s_1, \cdots, s_{n-1}, r_{0,1}, r_{1,0}, r_{1,2}, \cdots, r_{n-1,0}, r_{0,n-1})$$

ここで s_i は P_i の状態、r_{ij} はレジスタ R_{ij} に格納されている値を表す。 □

定義6 (ネットワーク状況の遷移) あるネットワーク状況 c がプロセッサの部分集合 U に属する各プロセッサがアルゴリズム $\mathcal{A}$ の原子動作を1回実行して状況が c' に変化した時、この状況の遷移を $c \xrightarrow{U} c'$ と記述する。どのプロセッサが動作したかを問題にしない場合は単に $c \rightarrow c'$ と記述する。さらに、状況 c から零回以上の遷移によって状況 c' が現れるとき、この遷移を $c \xrightarrow{*} c'$ と記述する。 □

定義7 (スケジュールとアルゴリズムの実行) ネットワークを構成するプロセッサの部分集合の無限列 $T = U_0, U_1, \cdots$ をスケジュールと呼ぶ。アルゴリズムを $\mathcal{A}$、初期状況 c_0 から始まるネットワーク状況の無限列を $E = c_0, c_1, \cdots$ とする。この時、E が全ての $i(\geq 0)$ について $c_i \xrightarrow{U_i} c_{i+1}$ を満たす時、E をアルゴリズム $\mathcal{A}$ の下でのスケジュール T の実行という。 □

定義 8 (実行の公平性) スケジュール Tにネットワークを構成する全てのプロセッサが無限回現れる時、Tは公平であるという。さらに Eがこのスケジュール Tの実行であるなら、この Eを公平な実行という。 □

2.3 自己安定

アルゴリズム $\mathcal{A}$ がある状況の集合Δに関して、以下の2つの条件を満足するならば、アルゴリズム $\mathcal{A}$ は正当な状況Δに関して自己安定であるという。

(1) 到達可能性 (No Livelock)
初期状況を c_0、アルゴリズムを $\mathcal{A}$ とする。初期状況 c_0 から $\mathcal{A}$ の公平な実行により $c_0 \xrightarrow{*} c_j$ かつ $c_j \in \Delta$ となる状況 c_j が現れる。

(2) 閉包性 (Closure)
任意の状況 $c \in \Delta$, c'に対し、$c \rightarrow c'$ならば $c' \in \Delta$が成立する。

3 リングの方向付けアルゴリズム

リングの方向付け問題とは、共通の方向感覚 (右または左) を持たないプロセッサ群に対して、共通の方向感覚を与える問題である。ここでは、奇数サイズのリングネットワークにおいてDデーモンの下で方向付け問題を状態数6で解く決定性アルゴリズムを提案し、その正当性の証明を行なう。

3.1 基本的なアイデア

リング中のすべてのプロセッサはその方向を表す内部変数 dirを持っている。同方向を向いているプロセッサの極大部分列をセグメントという (定義 11)。リングサイズは奇数である。したがって、もしリングが方向付けられていなければ、互いに向かい合うセグメントの組のうち、一方が奇数サイズで他方が偶数サイズであるような組が少なくとも1つ存在する (図1)。

このようなセグメントの組 (γ, γ') について考える。各セグメントに対して、図2のようにソース (定義 10) からラベルを $0, 1, 0, \cdots$と交互に付けてゆく。γおよびγ'のサイズはそれぞれ、奇数および偶数 (または偶数および奇数) であるから、γおよびγ'の先頭のプロセッサPとQのラベルはそれぞれ0と1(または1と0) となる。したがって、他のすべてのセグメントの組が対称な形 (つまり、図2のようなラベル付けで、各セグメントの先頭のラベルが共に0または1になっている) であっても、このセグメントの組にはそのようなラベル付けは生じない (図2)。

図2において、PとQのうち0でラベル付けされている方がその方向を変更する (つまり方向を逆にする) ことにすると、Pの方向はRの向きに変わる。しかし、Rのラベルが1かつPのラベルが0であるから、Pは再び方向を変えなければならない。以後この繰り返しが永久に続いてしまい、一方向に方向付けができない (livelock)。これを防ぐために、ラベルH(igh) を導入する。図2において、Pが方向を変えるとき、同時にそのラベルをHに変更する。このとき、Pのラベルは H、Rのラベルは1となる (図3)。図3において、PとRのうち1(または0) がラベル付けされている方がその方向を変更することにすると、Rの方向がPとは逆の向きに変わり、以後同様にしてセグメントを一方向に統一することができる。ところが、図2においてPとQのラベルが共にHの場合、P、Qは互いにその方向を変更しないために、再びlivelockが生じてしまう。しかし、この場合はラベルHを0(または1) に戻し、図2のような形を再構成することで解決することができる。

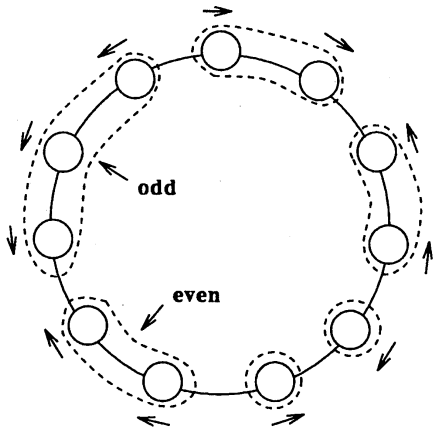

図 1: リングにおけるセグメント

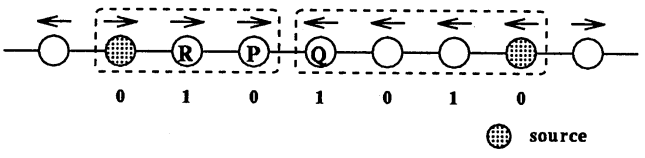

図 2: セグメントへのラベル付け

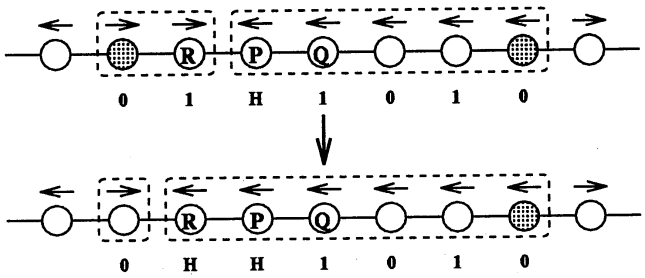

図 3: ラベル H の役割

3.2 アルゴリズム

各プロセッサの状態は、2 つの内部変数 $label = \{0, 1, H\}$ および $dir = \{F(orward), B(ackword)\}$ より構成される。各プロセッサ P_iにおいて、これらの内部変数の値はその隣接プロセッサに対するレジスタに格納される。図 4 に各プロセッサに対する状態遷移規則を示す。プロセッサ P_i が動作した時、P_iはその状態およびその第 1 隣接プロセッサ P_j の状態 (P_jのレジスタから読み込まれる) の組が図 4 の規則に合うものを探す。もし合致する規則が見つかれば、P_iはその規則に従って状態を変更する。合致する規則が見つからなければ、P_iの第 2 隣接プロセッサ P_kに対しても上記と同様の動作を行なう。その後、新しい状態をそのレジスタに書き込む。図 4 において、l_iは P_iの $label$の値を表しており、また $l_i := \neg l_j$は $l_j = 0(l_j = 1)$ であれば $l_i := 1(l_i := 0)$ であることを意味している。

3.3 正当性の証明

任意のプロセッサ P_iはアルゴリズム (付録参照) の原子動作を少なくとも 1 回実行すると、P_iでは以下の条件が成立する:

- $dir_i = F$の場合 $\cdots$ $r_{ij} = (l_i,\ B),\ \ r_{ik} = (l_i,\ F)$
- $dir_i = B$の場合 $\cdots$ $r_{ij} = (l_i,\ F),\ \ r_{ik} = (l_i,\ B)$

ここではアルゴリズムの公平な実行を仮定しているので、すべてのプロセッサにおいて上記の条件が成立するような状況 c_sが必ず現れる。以下の議論では c_s以降に現れる状況を仮定している。

定義 9 (各プロセッサの方向) 各プロセッサ P_iの状態は内部変数 dirおよび $label$から成る。dir_iを P_iの内部変数 dir とする。もし $dir_i = F$ならば P_iは第 2 隣接プロセッサ P_k の方向を、$dir_i = B$ならば第 1 隣接プロセッサ P_jの方向

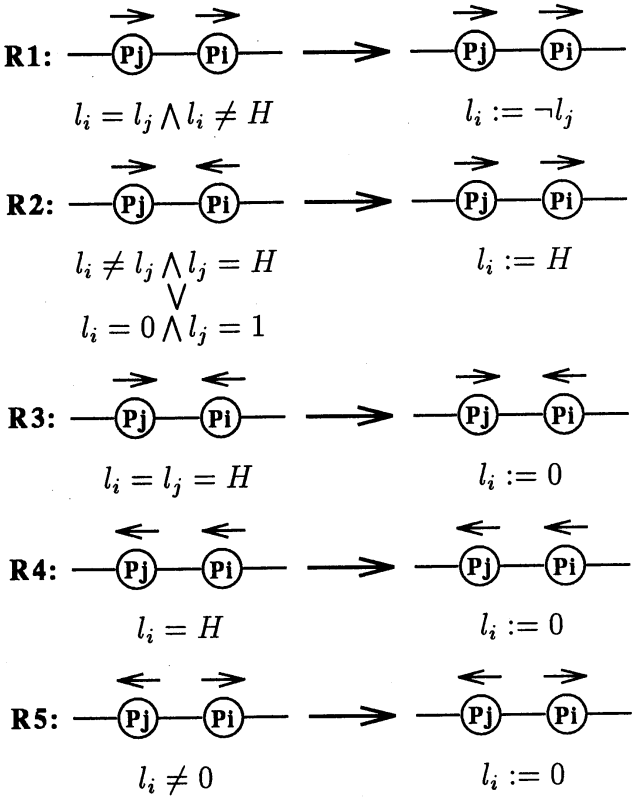

図 4: P_iに対する状態遷移規則

を向いているものとする。ここで、記述を簡単にするために、前者の場合は $head(P_i) = P_k$または $tail(P_i) = P_j$のように書く。後者の場合も同様である。 □

定義 10 任意のプロセッサを P_i とする。このとき、P_iにおいて

$$head(P_{i-1}) \neq P_i \ \wedge \ head(P_{i+1}) \neq P_i$$

が成立するとき、P_iをソース ($source$) という。 □

定義 11 以下のいずれかの条件を満たすプロセッサの極大部分列 $P_r, P_{r+1}, \cdots, P_s$をセグメント ($segment$) という。

1) $head(P_{r-1}) \neq P_r \wedge head(P_{s+1}) = P_s \wedge head(P_m) = P_{m+1}\ (r \leq m < s)$
2) $head(P_{s+1}) \neq P_s \wedge head(P_{r-1}) = P_r \wedge head(P_m) = P_{m-1}\ (r < m \leq s)$

ここで、1) のセグメントを順方向セグメント、2) のセグメントを逆方向セグメントと呼ぶ。 □

定義 12 $\gamma = P_r, P_{r+1}, \cdots, P_s$を任意の順方向セグメントとする。また$\gamma$に属する任意のプロセッサを $P_i(i = r, r+$

$1, \cdots, s)$ とする。このとき、γが以下の条件を満たすとき *well formed* であるという。

1. $l_r = 0$
2. $l_i = \neg l_{i-1} (i = r+1, \cdots, s-1)$
3. $l_s = \neg l_{s-1} \ \vee \ l_s = H$

γが逆方向セグメントである場合も同様である。 □

定義 13 (正当な状況Δ) ネットワーク状況 c において、任意のプロセッサ $P_i (i = 0, 1, \cdots, n-1)$ で次の条件が成立するとき、この状況 c は正当な状況にあるという。

$$tail(head(P_i)) = P_i$$

□

補題 1 任意の状況を c とする。もし $c \notin \Delta$ ならば、リング中にソースが少なくとも 1 つ存在する。

(**証明**) ソースが存在しないような状況 $c \notin \Delta$が存在すると仮定し、矛盾を導く。

任意のプロセッサを P_iとする。また、$head(P_i) = P_{i+1}$ とすると、P_iはソースではないから、P_{i-1}、P_{i+1}の方向は以下の 3 つの場合に分類できる:

(1) $head(P_{i-1}) = P_i \wedge head(P_{i+1}) = P_i$の場合
$head(P_i) \neq P_{i-1}$かつ P_{i-1}はソースではないから、$head(P_{i-2}) = P_{i-1}$である。また、$head(P_{i-1}) = P_i$ かつ P_{i-2} はソースではないから $head(P_{i-3}) = P_{i-2}$ となる。以下同様にして $head(P_{i+3}) = P_i$(つまり $head(P_{i+1}) \neq P_{i+2}$) であるから、$P_{i+2}$はソースとなり仮定に矛盾する。

(2) $head(P_{i-1}) = P_i \wedge head(P_{i+1}) \neq P_i$の場合
(1) の場合と全く同様にすると、任意の m について $head(P_m) = P_{m+1}$となる状況が得られる。この状況は正当な状況に含まれるので考えなくて良い。

(3) $head(P_{i-1}) \neq P_i \wedge head(P_{i+1}) = P_i$の場合
$head(P_i) = P_{i+1}$であるから、P_{i+2}の方向に関係なく P_{i+1}はソースとはならない。また、$head(P_{i+1}) = P_i$ であり、P_{i+2}はソースではないから、$head(P_{i+3}) = P_{i+2}$となる。リングサイズが奇数であることに注意して、以下同様にすると $head(P_{i-2}) = P_{i-3}$を得る。ところが、$head(P_i) \neq P_{i-1}$であるから、P_{i-1}はソースとなり仮定に矛盾する。

$head(P_i) = P_{i-1}$の場合も上記と全く同様にして示すことができる。 □

補題 2 : アルゴリズム A はΔに関して閉包性を満たす。

(**証明**) 任意のプロセッサを P_i、ある正当な状況を $c \in \Delta$ とする。状況 c において、補題 1 よりソースは存在せず、すべてのプロセッサは同方向 (共通の右または左) を向いている。P_iが動作すると R1 または R4 以外の規則が適用されることはなく、R1 または R4 の適用により動作後の状況でその方向が変化することはない。以上のことは複数のプロセッサが同時に動作しても成立する。 □

定義 14 : 任意の状況 c とする。状況 c において、リング中に存在するセグメントの数を $z(c)$ と記述する。ここで、$z(c)$ は $0 \leq z(c) \leq n-1$ を満たし、かつ偶数であることに注意せよ。 □

補題 3 : 任意の状況を c とする。このとき状況 $c' (c \xrightarrow{*} c')$ において $z(c) \geq z(c')$ が成立する。 □

(**証明**) アルゴリズムの適用によりセグメント数が増加しないことは容易に分かる。よって、本補題が成立する。 □

定義 15: 任意のアルゴリズムの実行を $E = c_0, c_1, \cdots$とし、Eのある部分実行を $E' = c_i, c_{i+1}, \cdots$とする。$E'$中に現れる任意の状況 $c_j (j \geq i)$ について $z(c_i) = z(c_j)$ が成立するとき、実行 E'は *quiet* であるという。 □

補題 4 : 実行 $E' = c_i, c_{i+1}, \cdots$が *quiet* であるとする。状況 c_iにおいてリング中に存在する任意のセグメントをγとすると、E'中にγが *well formed* となるような状況 c_wが現れる。

(**証明**) E'中に現れるある状況を c とし、また任意のセグメントを$\gamma = P_r, P_{r+1}, \cdots, P_{g(c,\gamma)}$とする。ここで、$g(c, \gamma)$ は状況 c におけるγの先頭のプロセッサの添字を表している。以下の主張を証明することにより、本補題が成立することを示す。

主張 1: 状況 c において、P_rが動作すると動作後の状況 c' では $l_r = 0$ が成立する。

主張 2: 状況 c において、$\gamma' = P_r, P_{r+1}, \cdots, P_m$ が *well formed* かつ P_{m+1}がγに属しているとする。このとき、状況 c 以降に現れる状況において P_{m+1}がγに属しているならば、γの部分列$\gamma'' = P_r, P_{r+1}, \cdots, P_{m+1}$も *well formed* となる状況 c'が現れる。

(**主張 1 の証明**) P_rがソースならばR5のみが適用されるので明らかである。P_rがソースでない場合、P_rに対して考えられる規則はR2、R3およびR5である。R2が適用されるとγは消滅してしまい、E'が *quiet* であることに反するから、この場合は考えなくても良い。R3またはR5が適用された場合、動作後の状況c'では$l_r = 0$となる。

(**主張 2 の証明**) 部分列γ'は *well formed* であるから、$l_m = H$の場合も考えられる。しかし、状況c以降に現れる状況では$head(P_{m+1}) \neq P_m$が成立するので、P_mが少なくとも2回動作するとR1およびR4により$l_m = \neg l_{m-1}$となる状況が現れる。したがって、以下の議論では$l_m \neq H$の場合について考える。

(1) $r < m < g(c,\gamma) - 1$の場合:
P_{m+1}に対して考えられる規則はR1およびR4である。したがって、少なくとも2回のP_{m+1}の動作後の状況では$l_{m+1} = \neg l_m$となり、部分列$\gamma'' = P_r, P_{r+1}, \cdots, P_{m+1}$は *well formed* となる。ここで、$P_{m+1}$と同時またはそれ以前に$\gamma'$に属するプロセッサが動作してもそれらの状態は変化しないことに注意せよ。

(2) $m = g(c,\gamma) - 1$の場合:
P_{m+1}はγの先頭のプロセッサであり、またγは *well formed* ではないから$l_{m+1} \neq H$である。したがって、P_{m+1}に対して考えられる規則はR1およびR2である。P_{m+1}にR2が適用されると動作後の状況ではP_{m+1}はγに属さないので、この場合は考えなくても良い。R1が適用された場合、動作後の状況c'では$l_{m+1} = \neg l_m$となり、部分列$\gamma'' = P_r, P_{r+1}, \cdots, P_{m+1}$は *well formed* となる。

以上より本補題が成立する。 □

補題5: アルゴリズムAはΔに関して到達可能性を満たす。

(**証明**) ある状況をcとする。もし$c \in \Delta$ならばリング中にセグメントは存在しないので$z(c) = 0$となり、$c \notin \Delta$ならば$z(c) > 0$である。また、補題3より正当な状況が現れないと仮定すると以下のことが成立する:

$$\exists t(\geq 0), \exists k(\in \{2,4,\cdots\}), \forall m(>0)\ [z(c_m) = z(c_t) = k]$$

ここで、c_tは状況c_sからt回の遷移で、c_mは状況c_tからm回の遷移の後に現れる状況である。以下にこれは矛盾であることを示す。

状況c_tにおいて、リング中に存在するセグメントの組のうち、サイズが偶数と奇数の組(γ,γ')について考える。実行$E' = c_t, c_{t+1}, \cdots$は仮定より *quiet* である。補題4より$\gamma$および$\gamma'$が共に *well formed* となるような状況c_wがE'中に現れる。状況c_wにおいて一般性を失うことなく、γは順方向セグメントかつそのサイズが偶数である(リングサイズnは奇数よりγ'のサイズは奇数である)と仮定できる。ここで、γの先頭のプロセッサをP_i(γ'の先頭のプロセッサはP_{i+1})とすると、P_iとP_{i+1}の間で以下の関係式のいずれかが成立する:

a. $l_i = 1 \ \wedge \ l_{i+1} = 0$

b. $l_i = H \ \wedge \ l_{i+1} = 0$

c. $l_i = 1 \ \wedge \ l_{i+1} = H$

d. $l_i = H \ \wedge \ l_{i+1} = H$

(a が成立している場合)
状況c_wにおいてP_{i+1}が動作すると(他のプロセッサが動作しても状況は変化しないことに注意)、R2が適用されて動作後の状況ではP_{i+1}、P_{i+2}がそれぞれγおよびγ'の先頭となり、動作後の状況ではP_{i+1}とP_{i+2}の間で以下の関係式が成立する:

$$l_{i+1} = H \ \wedge \ l_{i+2} = 1 \ \wedge \ head(P_{i+1}) = P_{i+2}$$

このとき、動作後の状況が変化するのはP_{i+2}のみである。P_{i+2}が動作すると、R2が適用されて動作後の状況ではP_{i+2}とP_{i+3}の間で以下の関係式が成立する:

$$l_{i+2} = H \ \wedge \ l_{i+3} = 0 \ \wedge \ head(P_{i+2}) = P_{i+3}$$

以下同様にして、γ'がγに吸収されて$z(c) = 0$となる状況$c \in \Delta$が現れる。また、状況c_wにおいてbまたはcが成立している場合も同様にして示すことができる。これは仮定に矛盾する。

(d が成立している場合)
状況c_wにおいて、動作後の状況が変化するのはP_iまたはP_{i+1}が動作した場合のみである。P_iとP_{i+1}に対して同時にR3が適用された場合、動作後の状況ではP_iとP_{i+1}の間で以下の関係式が成立する:

$$(l_i = 1 \ \wedge \ l_{i+1} = 0) \ \vee \ (l_i = 0 \ \wedge \ l_{i+1} = 0)$$

前者の場合はaと同じ状況である。後者の場合、さらにP_iが動作すると(ここで、他のプロセッサが動作しても状況は変化しないことに注意)、動作後の状況では$l_i = 1 (l_{i+1} = 0)$となる。したがって、以下aの場合と同様にして示すこと

ができる。また、状況 c_w において $P_{i+1}(P_i)$ にのみ R3 が適用された場合、動作後の状況では P_i と P_{i+1} の間で次式が成立する:

$$l_i = H \ \wedge \ l_{i+1} = 0 \ ((l_i = 0 \vee l_i = 1) \ \wedge \ l_{i+1} = H)$$

この場合も上記と同様の議論ができる。これは仮定に矛盾する。以上より本補題が成立する。 □

定理 1: アルゴリズム A は奇数サイズのリングにおいて、D デーモンの下で方向付け問題を状態数 6 で解く自己安定アルゴリズムである。

(証明) 補題 2 および補題 5 より成立する。 □

4 おわりに

本稿では、均一な奇数サイズのリングネットワークにおいて D デーモンの下で方向付け問題を状態数 6 で解く自己安定アルゴリズムを提案した。これまでに、奇数サイズのリングネットワークにおいて D デーモンの下で無限状態数で解く決定性アルゴリズムが知られていたが、本稿では状態数 6 で解けるというより強い結果を示した。リングの方向付け問題の状態数の下限および R/W デーモンの下で定数状態数で解けるか否かについては未解決であり興味ある問題である。

参考文献

[1] T. Herman, "Probabilistic Self-Stabilization," *Infomation Processing Letters*, Vol.35, pp.63-67(1981).

[2] J.E. Burns, J. Pachl, "Uniform Self-Stabilizing Rings," *ACM Transactions on Programming Languages and Systems*, Vol.11, No.2, pp. 330-344(1989).

[3] 片山喜章, 増澤利光, 都倉信樹, "リングの方向付け問題を解く自己安定アルゴリズム," 情処研報, アルゴリズム 25-8(1992).

[4] S. Dolev, A. Israeri, "Uniform Dynamic Self-Stabilizing Leader Election," *Proceedings of the 5th International Workshop on Distributed Algorithms(LNCS 486)*, pp.31-51(1991).

[5] A. Israeri, M. Jalfon. "Self-Stabilizing Ring Orientation," *Proceedings of the 4th International Workshop on Distributed Algorithms (LNCS 486)*, pp.1-13(1990).

[6] E.W. Dijkstra, "Self-Stabilizing Systems in Spite of Distributed Control," *Communications of the ACM*, Vol.17, No.11, pp.643-644(1974).

付録

```
do forever
begin  /* 原子動作の開始 */
   /* 隣接レジスタからの値の読み込み */
   sr1 := read(rji); sr2 := read(rki);

   if sr1.dir = F then
      if dir = F and label = sr1.label and label ≠ H then
         label := ¬sr1.label;
      elseif dir = B then
         if (label ≠ sr1.label and sr1.label = H)
               or (label = 0 and sr1.label = 1) then begin
            label := H; and dir := F;
            end
         elseif label = sr1.label and label = H then
            label := 0;
    elseif sr1.dir = B and ((dir = B and label = H)
       or (dir = F and label ≠ 0)) then
      label := 0;

    if sr2.dir = F then
      if dir = B and label = sr2.label and label ≠ H then
         label := ¬sr2.label;
      elseif dir = F then
         if (label ≠ sr2.label and sr2.label = H)
               or (label = 0 and sr2.label = 1) then begin
            label := H; and dir := B;
            end
         elseif label = sr2.label and label = H then
            label := 0;
    elseif sr2.dir = B and ((dir = F and label = H)
       or (dir = B and label ≠ 0)) then
      label := 0;

   /* 隣接レジスタへの値の書き込み */
   if dir = F then begin
      write rij := (label, B); write rik := (label, F);
      end
   else begin
      write rij := (label, F); write rik := (label, B);
      end
end  /* 原子動作の終了 */
```

リングの方向付けアルゴリズム